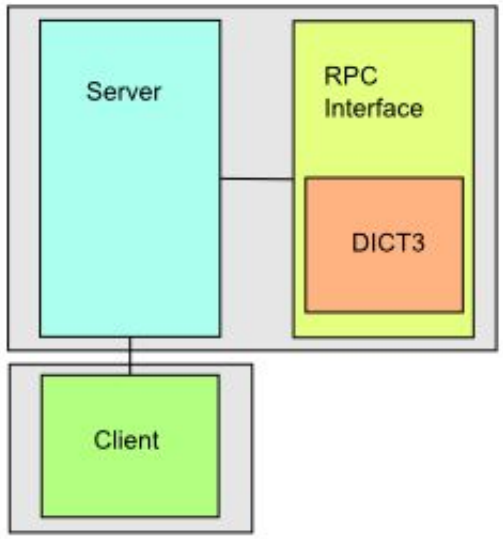
Grayson Hardaway
CMSC 621
Dr. Kalpakis
March 3, 2015

# Project 1 Design

This project implements a simple RPC server for a look-up system using the JSON-RPC library available in Go.

The major components of this project are:

1. The DICT3 library;
2. The RPC interface for the DICT3 library;
3. The server, which facilitates network connections and interacts with the DICT3's RPC interface; and
4. The client.




**DICT3**

The DICT3 library supports a look-up data structure which uses two keys--one actually called a *key* and the other a *relationship.* The look-up value can be any generic structure.

In code, this is accomplished by creating a **KeyRelationship** structure, which contains two strings. Combining these into a single structure facilitates easy look-up into a standard Go **map[]**. The value of the map is an **interface{}**. This structure--**map[KeyRelationship]interface{}**--is called **DICT3**.

The DICT3 library defines a series of methods which take a pointer to a DICT3 structure as a receiver. These methods precisely mirror the function signatures in the project outline: **Lookup, Insert, InsertOrUpdate, Delete, ListKeys, ListIDs, and**

**Shutdown**. (In Go, a capital letter indicates the property is public.) Furthermore, the DICT3 library implements two methods, **Save** and **Load**, which save and load respectively the contents of the DICT3 in memory to or from a file.

The file format uses lines as entries in the DICT3; each line has three fields: the key, the relationship, and the value. Fields are pipe-delimited ( | ). In the file, the value is serialized as a JSON object. Example: KEY|REL|{“a”:[1,2,3]}

**RPC Interface**

Go provides libraries for implementing RPC clients and servers, including one for encoding using JSON. Go’s RPC libraries operate by *registering* an initialized structure, and then examining methods on that structure for a specific signature:

```
func (t *T) MethodName(argType T1, replyType *T2) error
```

The RPC interface wraps the DICT3 library so that its methods take the above form, allowing registration with the RPC library.

**Server**

The server handles initialization and registration of a DICT3 object before establishing a connection for listening for incoming JSON RPC requests. As requests come in, the server dispatches goroutine requests to the RPC library which answers the requests and returns the responses.

When initialized, the server reads in a configuration file stored in JSON form which contains relevant information such as its port number.

**Client**

The client simply reads input from StandardIn and sends it to the server. It is expected that input will be valid JSON RPC strings; anything which does not match the specification will be rejected by the server.

The client also uses the same JSON server configuration file to establish successful communication with the server.

**Running the Server/Client**

The project directory contains two shell scripts, **client.sh** and **server.sh**, which start the client and server respectively. In order to run correctly, these scripts must be run from the present working directory (e.g., **./server.sh**). Both scripts take the server configuration file as an argument, which is currently in the *config* directory.

**Supported Client Methods**

Due to the RPC interface wrapping the DICT3 library, the RPC method names differ slightly from the original DICT3 methods. Methods in the JSON RPC requests take the form **DICT3.R<methodname>**, where DICT3 is the type of object registered by the RPC library, and the capital "R" stands for "remote." Furthermore, the parameters to the methods are named, and therefore must be contained within a JSON object.

DICT3.RLookup

```
{
    "method":"DICT3.RLookup",
    "params":[{"key":"<key>", "relationship":"<relationship>"}],
    "id":1
}
```

DICT3.RInsert

```
{
    "method":"DICT3.RInsert",
    "params":[{"key":"<key>", "relationship":"<relationship>",
        "value":<JSONObject>}],
    "id":1
}
```

DICT3.RInsertOrUpdate

```
{
    "method":"DICT3.RInsertOrUpdate",
    "params":[{"key":"<key>", "relationship":"<relationship>",
        "value":<JSONObject>}],
    "id":1
}
```

DICT3.RDelete

```
{
    "method":"DICT3.RDelete",
    "params":[{"key":"<key>", "relationship":"<relationship>"}],
    "id":1
}
```

DICT3.RListKeys

```
{
    "method":"DICT3.RListKeys",
    "params":[],
    "id":1
}
```

DICT3.RListIDs

```
{
    “method”:“DICT3.ListIDs”,
    “params”:[],
    “id”:1
}
```

DICT3.RShutdown

```
{
    “method”:“DICT3.RShutdown”,
    “params”:[],
    “id”:1
}
```